NOVA SCHOOL OF SCIENCE & TECHNOLOGY
DEPARTAMENTO DE INFORMÁTICA

Arquitetura de Computadores 2024/25

TPC 2

Entrega: 18h00 de 15 de abril de 2025

Este trabalho de casa consiste num exercício de programação a ser realizado em grupo de no máximo dois alunos. Pode esclarecer dúvidas gerais com colegas, mas a solução e a escrita do código deve ser estritamente realizada pelos membros do grupo. Os casos de plágio serão punidos de acordo com os regulamentos em vigor.

A forma de entrega será através do Mooshak, usando os identificadores do trabalho anterior, a menos que o grupo tenha sido alterado. Informação mais detalhada será enviada através de uma mensagem via CLIP.

Exercício

Neste exercício deve completar o código de um simulador (escrito na linguagem C) de uma arquitetura composta por um CPU muito simples e uma memória. Este inclui, já implementada, uma consola (ficheiro `sim.c`) onde é possível um utilizador dar comandos para ler e escrever na memória central, incluindo ler para memória os valores representados num ficheiro de texto. Pode ainda mandar executar as instruções presentes na memória. Os comandos implementados são os seguintes:

> `poke` *EEE X* – escreve no endereço *EEE* o valor *X*.
>
> `peek` *EEE* – ler o valor no endereço de memória *EEE*. O valor é afixado em hexadecimal.
>
> `dump` *X* – mostra todo o conteúdo do acumulador e de *X* palavras da memória a partir do endereço 0.
>
> `load` *FILE* – lê o ficheiro de nome *FILE*, interpretando cada linha de texto como um valor que é colocado sequencialmente na memória.
>
> `run` – executa as instruções a partir do endereço 0 de memória.

Os valores *X* e *EEE* podem ser dados em base 10 ou em base 16 se iniciados por `0x`.

O código a completar está no ficheiro `dorun.c` e deve implementar o ciclo de *fetch*, *decode* e *execute* para simular a execução das instruções na memória. A execução deve decorrer sempre a partir do endereço 0 de memória, até à execução da instrução HALT (ver secção seguinte).

Instruções máquina do processador:

O processador usa palavras de 16 bits e tem 1 registo de uso geral chamado acumulador registos (AC). Tem ainda um registo *Program Counter* (PC) e outro para a instrução a executar (IR). A memória está organizada em palavras de 16 bits por endereço, sendo cada endereço de 12 bits.

As instruções têm tamanho fixo de 16 bits, sendo os 4 mais significativos para o código de operação e os restantes 12 para o endereço.

15 12 11 0

Código da Operação (4 bits)	Endereço (12 bits)

Bits 15-12: Código da instrução: especifica a instrução a executar e a interpretação dos bits 11 a 0

Bits 11-0: endereço: especifica o endereço de memória a usar nas instruções que a referenciam

Há exceções a esta organização e em algumas instruções há bits que não são considerados. As instruções suportadas e o respetivo código máquina em representação hexadecimal, são descritas a seguir. São usadas as convenções:

Símbolo	Significado
eeeeeeeeeeee	Endereço com 12 bits; é um inteiro sem sinal
xxxxxxxxxxxx	Bits a ignorar
vvvvvvvvvvvv	12 bits que especificam um valor codificado em complemento para 2 ou seja um inteiro com sinal que pode representar valores de 2^{11} -1 a -2^{11}

Instrução	**Mnemónica**	**Descrição**
0000xxxxxxxxxxx	HALT	Termina o programa
0001eeeeeeeeeee	LOAD endereço	O registo acumulador recebe o valor contido na posição de memória cujo endereço é *eeeeeeeeeee*
0010eeeeeeeeeee	STORE endereço	O valor guardado no acumulador é colocado na posição de memória com endereço *eeeeeeeeeee*
0011eeeeeeeeeee	ADD endereço	O conteúdo da posição de memória com endereço *eeeeeeeeeee* é somado ao conteúdo do acumulador e o resultado é guardado no acumulador
0100eeeeeeeeeee	SUB endereço	O conteúdo da posição de memória com endereço *eeeeeeeeeee* é subtraído ao conteúdo do acumulador e o resultado é guardado no acumulador
0101eeeeeeeeeee	MUL endereço	O conteúdo da posição de memória com endereço *eeeeeeeeeee* é multiplicado pelo conteúdo do acumulador e o resultado é guardado no acumulador
0110eeeeeeeeeee	DIV endereço	O conteúdo do acumulador é dividido pelo conteúdo da posição de memória com endereço *eeeeeeeeeee* e o resultado é guardado no acumulador
0111eeeeeeeeeee	JUMP endereço	O PC recebe o valor *eeeeeeeeeee* que está nos bits 11 a 0.
1000eeeeeeeeeee	JZ endereço	O PC recebe o valor *eeeeeeeeeee* que está nos bits 11 a 0, se o conteúdo do acumulador é 0.
1001eeeeeeeeeee	JN endereço	O PC recebe o valor *eeeeeeeeeee* que está nos bits 11 a 0, se o o bit mais significativo do acumulador for 1, o que quer dizer que o conteúdo é negativo
1010eeeeeeeeeee	CALL endereço	O PC corrente mais um é guardado no endereço *eeeeeeeeeee* que está nos *bits 11 a 0* e o PC recebe *eeeeeeeeeee* mais um.
1011eeeeeeeeeee	RETURN endereço	O PC recebe o valor que está guardado na posição de memória com endereço *eeeeeeeeeee.*
1100vvvvvvvvvvv	LDI vvvvvvvvvvv	O acumulador recebe o valor *vvvvvvvvvvv*. Como o acumulador tem 16 bits e vvvvvvvvvvv só tem 12, faz-se a extensão do sinal. Se v_{11} é 0, o acumulador recebe 00000vvvvvvvvvvv; se v_{11} é 1, o acumulador recebe 11111vvvvvvvvvv.
1101	Código não utilizado	
1110	Código não utilizado	
1111xxxxxxxxxxx	NOP	Instrução que não faz nada.

Exemplo de um programa que calcula o valor de y = 2^x através da fórmula de recorrência. O resultado *y* é colocado na posição 0x11; x é o valor que está na posição 0x10 e na posição 0x12 está o valor 1.

```
end.   code     mnemónica
0x000:  0x1012   LOAD   0x012    // AC ← 1
0x001:  0x2011   STORE  0x011    // y ← 1
0x002:  0x1010   LOAD   0x010    // AC ← x
0x003:  0x800F   JZ     0x00F    // termina o programa saltando para 0x00F
0x004:  0x4012   SUB    0x012    // AC ← AC - 1
0x005:  0x2010   STORE  0x010    // x ← AC
0x006:  0x1011   LOAD   0x011    // AC ← y
0x007:  0x3011   ADD    0x011    // AC ← AC + y
0x008:  0x2011   STORE  0x011    // y ← AC (em que AC = 2 * y)
0x009:  0x7002   JMP    0x002    // inicia um novo ciclo
0x00A:  0xF000   NOP
0x00B:  0xF000   NOP
0x00C:  0xF000   NOP
0x00D:  0xF000   NOP
0x00E:  0xF000   NOP
0x00F:  0x0000   HALT
0x010:  0   (x  preeenchido com poke)
0x011:  0   (y para ser lido no final com peek)
0x012:  1   (para decrementar)
```

O ficheiro `p.code` fornecido contém o código e dados deste programa em notação hexadecimal, ocupando uma palavra de memória por linha. Correndo o simulador, podemos carregar o programa fazendo `load p.code` e executar com `run`. Podemos consultar o resultado vendo o que fica na variável de endereço 0x11 com `peek`. Podemos alterar o valor de *x* usando o comando `poke`. Exemplo duma sessão para calcular *2^5*:

```
cmd> load p.code
cmd> poke 0x10 5
cmd> run
HALT instruction executed
cmd> peek 0x11
0x11: 0x20
```

Entrega

A entrega faz-se através do sistema Mooshak; deve ser feita dentro do prazo, submetendo apenas o seu ficheiro *dorun.c.*

O programa será compilado com o seguinte comando:

```
gcc -Wall -std=c11 -o sim sim.c dorun.c
```